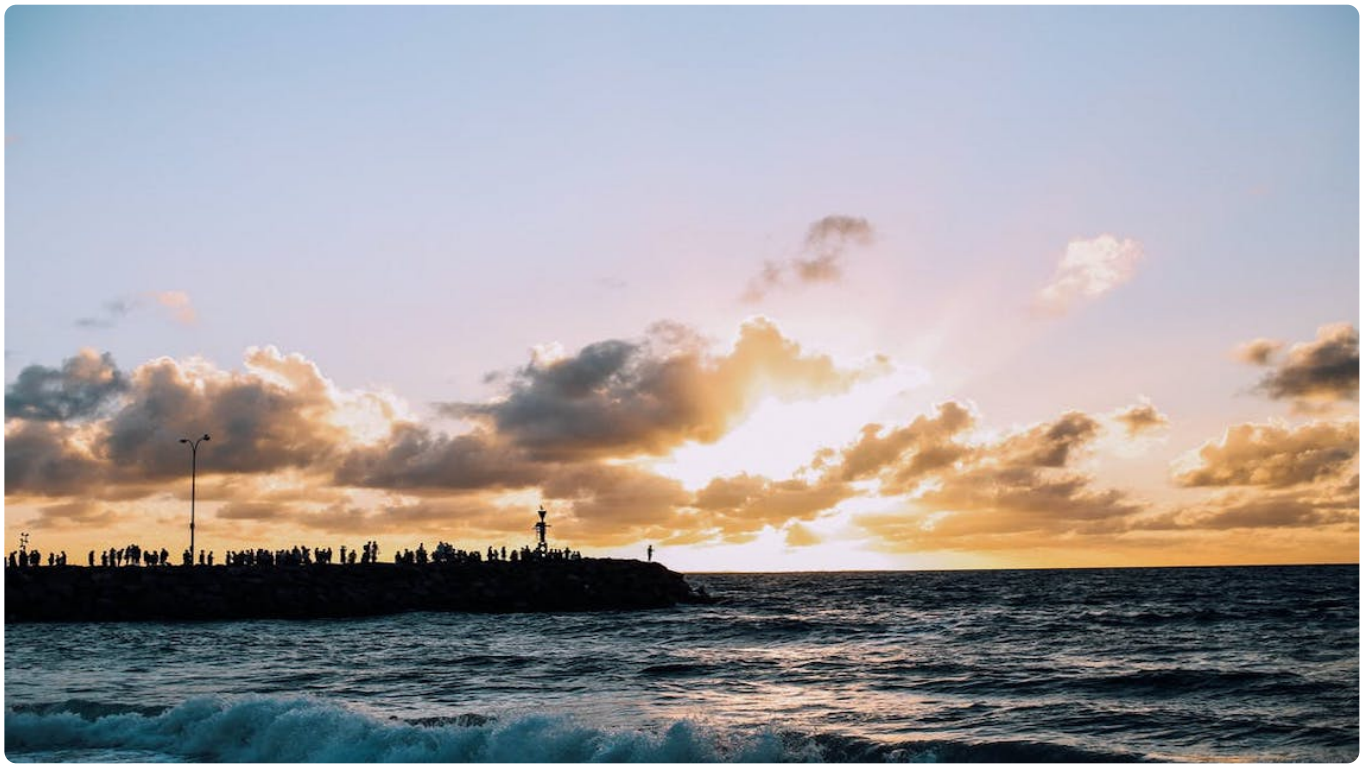
04 | 函数与字典:如何实现多次替换

2021-02-18 尹会生

Python自动化办公实战课 进入课程 >



讲述：尹会生
时长 13:47 大小 12.64M

你好，我是尹会生。

“替换”是我们日常办公经常遇到的操作，比较常见是把文件中的英文标点符号（,.""”）替换成中文标点符号（，。 “”）。有时候不仅是标点符号，还需要替换好几个词。还有一种情况不太常见，但是一碰到就很棘手，那就是根据数字范围进行替换，比如“0-6岁”替换成“少年”，“7-17 岁”替换成“青年”，“18-65 岁”替换成“中年”。

如果直接使用替换函数，你需要编写大量的代码逻辑，但是使用逻辑判断和字典就可以用更高效的代码来实现快速替换功能。那么今天这节课，我们就来学习下怎么实现快速替换。

用 Python 实现“替换”功能的原理

为了让你更直观地理解编程语言里的替换，我先来给你讲一讲用 Python 实现替换的原理。我用一个例子来给你讲解。比如我需要把字符串“新年快乐”替换为“恭喜发财”，在 Python 中，我是通过 replace() 函数来实现的：

复制代码

```
1 string1="aaa新年快乐bbb"
2 string2=string1.replace("新年快乐", "恭喜发财")
3 print(string2)
4 # aaa恭喜发财bbb
5
6 string3="aaa新年快乐bbb新年快乐ccc"
7 string4=string3.replace("新年快乐", "恭喜发财", 2)
8 print(string4)
9 # aaa恭喜发财bbb恭喜发财ccc
```

你可以看到，在这段代码中我使用了 replace() 函数来实现文件内容的替换。为什么使用的是字符串的替换函数呢？因为在编程语言中，我们通常会把文件内容读取到内存用变量临时储存，再进行处理。为了便于对文字进行查找替换这类的操作，通常会使用字符串这种数据类型的变量来存储文字内容。

实现字符串替换的 replace() 函数，是 Python 字符串内置的函数。它既可以实现匹配关键字进行一次替换，也能支持对多次出现的字符串进行替换。比如在代码的第 7 行我就增加了参数“2”，这就实现了“新年快乐”两次的替换。

刚刚说的 replace() 函数，实现字符串替换操作的原理，就是从字符串中从前向后, 逐个字符去和 replace() 函数的第一个参数去比较, 如果文字内容相同，就把匹配的内容替换为 replace() 函数第二个参数的字符串内容。如果不匹配, 就继续对比下一个字符, 直到整个字符串比对完成. 为了便于描述，我把这种逐一比对, 找到相同字符进行替换的操作, 称作“一对一替换”。

在实际工作中，你遇到的替换场景会更复杂。比如需要把字符串中出现的所有英文标点符号，全部一对一地替换成中文的标点符号。那么这个问题其实就变成了多个“一对一”的替换操作了。该怎么解决呢？

相信你会想到通过多个 replace() 替换函数来实现多个“一对一”替换操作。为了让你更好地理解这一逻辑，我使用了如下代码，给你演示一下它的基本功能。

复制代码

```
string5='aaa,."bbb'
string6=string5.replace(',', ', ')
string6=string6.replace('.', '。')
string6=string6.replace('"', '“')
# 需要更多的replace()匹配更多的标点符号
print(string6)
# aaa, 。“bbb
```

在这段代码当中，我使用了三个 replace() 函数实现“,."” 三个符号的替换。不过一旦考虑替换更多的符号时，就要编写更多个 replace() 函数。这一行为虽然不会在运行效率上产生问题，但是会带来代码阅读上的障碍。

比如需要你通过 Python 把全国的省市地县的汉语拼音替换成汉字，像把“GUANGDONG”替换成“广东省”，你至少要编写上百个替换函数。那面对如此大量的“一对一”替换，我们该怎么高效地编写代码呢?

怎样实现批量替换?

我来解决这类问题一般会采用两种方式实现，一种方式是用字典 + 自定义函数替代 replace() 函数，另一种是用逻辑判断 + 自定义函数替代 replace() 函数。我们先来看字典 + 自定义函数的方式是怎么对带有大量 replace() 的程序进行优化的。

用字典 + 自定义函数替代 replace 函数实现批量“一对一”替换

我们还是用把城市名称的拼音替换成汉字的例子来讲解。为了让你更直观地比较字典 + 自定义函数方式和 replace() 函数的区别，我先给你演示一下实现替换功能的代码：

复制代码

```
# 保存映射关系的函数,函数的主要功能是通过字典实现的
def replace_city(city_name):
    return {
        "GUANGDONG":"广东省",
        "HEBEI":"河北省",
        "HUNAN":"湖南省",
        "HANGZHOU":"杭州市"
    }[city_name]

# 根据映射关系实现批量循环
def replace_multi(my_citys, replaced_string):
    for pinyin_city in my_citys:
```

```
        replaced_string = replaced_string.replace(
            pinyin_city,replace_city(pinyin_city))
    return replaced_string

# 哪些城市要替换
citys = ("GUANGDONG", "HUNAN")

# 需要替换的字符串
string1 = """
GUANGDONG，简称“粤”，中华人民共和国省级行政区，省会广州。
因古地名广信之东，故名“GUANGDONG”。位于南岭以南，南海之滨，
与香港、澳门、广西、HUNAN、江西及福建接壤，与海南隔海相望。"""

string2 = replace_multi(citys, string1)
print(string2)
# 广东省，简称“粤”，中华人民共和国省级行政区，省会广州。
# 因古地名广信之东，故名“广东省”。位于南岭以南，南海之滨，
# 与香港、澳门、广西、湖南省、江西及福建接壤，与海南隔海相望。
```

我在代码里是通过两个核心函数来实现替换的，它们分别是 replace_city() 和 replace_multi() 函数。

我们先来分析一下 replace_city() 函数。它实现的是城市拼音和城市中文名称的全部对应关系，其中有两个技术细节需要你掌握。

第一个技术细节是字典的取值方式。如果我把一个字典定义为 dict1，并且想取得字典的值，就可以使用这样的代码：

复制代码

```
dict1["abc"]=123
```

方括号中的字符串"abc"被称作字典的下标。通过下标，我们可以获得字典的值。为了定义字典以后可以反复使用，通常我们会给字典赋予一个变量名，以此作为字典名称。所以在这里 dict1 就是字典的名称。

当然，如果字典只使用一次，那也可以不使用字典名称。相应的，它的写法就变成了这样：

复制代码

```
1 {"abc":123, "aaa":456}["abc"]
```

通过这一行代码，你可以取出直接使用字典的值，而不需要对字典进行声明，也不需要为字典再起一个变量名。

第二个技术细节是我为 replace_city() 增加了一个参数 city_name，以及一个关键字 return。city_name 作为城市的拼音传入函数后，会作为字典的 key，通过字典的映射功能得到中文城市名称。而 return 关键字返回字典映射的结果，就是城市的中文名称。

通过这两个技术细节，就可以让函数 replace_city() 实现接收拼音并返回中文的功能。这样实现映射关系的好处是：函数调用一次就返回一个值，编写好这类函数之后，其他人可以拿去直接使用，不用考虑函数内部使用了哪种数据类型，有利于代码的重复使用。

除了 replace_city() 之外，还有一个核心函数 replace_multi() 函数，它通过 for 循环来实现批量“一对一”的替换。它的作用是避免重复编写大量的 replace() 函数，提高代码的可读性。

我在设计 replace_multi() 函数的时候，为它准备了两个参数。第一个是要替换的城市的拼音，第二个是要替换的字符串。

第一个参数我具体指定了哪些城市需要替换，这样编写会让我的程序更加灵活，不必把所有城市的拼音都进行拼音到中文的替换操作。

第二个参数也是为了让 replace_multi() 函数更加灵活，如果对多段文字进行替换，可以多次调用 replace_multi() 函数。同样的，如果 replace_multi() 函数需要多次调用，也可以通过循环结构批量来优化代码。

总结来说，通过字典 + 自定义函数替代字符串默认的替换函数 replace() 函数，可以避免编写大量的 replace() 函数，提高了代码的灵活性和可读性。如果你在工作中涉及这类大量的“一对一”替换时，可以考虑采用我教的这个方法来优化你的替代效率。

用逻辑判断 + 自定义函数替代 replace() 函数实现“多对一”替换

除了刚才我提到的大量“一对一”的替换场景，还有一种替换场景你也会遇到，并且一旦遇到就很棘手。

比如在 Excel 中，你需要根据年龄这一列单元格来把你的客户划分为少年、青年、中年、老年。如果把年龄的每个整数都进行一次替换，这种写法会非常啰嗦，所以我们可以使用逻辑判断来实现这一替换。我先把代码给你演示出来。

复制代码

```
age = 18
if age>0 and age<=6:
    value="少年"

elif age>7 and age<=18:
    value=青年"

elif age>19 and age<=65:
    value="中年"

else:
    value="老年"
```

这段代码通过逻辑判断实现了从年龄到少年、青年等年龄段的替换功能。在代码中，“if” “elif” “else”是构成逻辑判断的关键字，它们表示了如果关键字到“:”之间的结果为 True，则其他语句后面的代码不会被执行。

根据匹配的年龄要求，年龄是一个范围，所以我使用了 and 关键字连接两个判断逻辑。比如“age>0 and age<=6” 代码，意思就是当 age 同时满足大于 0，并且小于等于 6 时，判断的条件才成立，这段代码的返回结果为 True，因此 value 变量的值就是“少年”。

在这段代码中我使用了一个逻辑判断结构。逻辑判断结构用于判断 age 变量的范围，它可以根据判断的结果为 value 变量进行赋值。这种实现形式和城市的拼音替换不同，城市的拼音和汉字是逐一对应的,age 变量的多个值, 例如从 1 到 6 对应的都是“少年”, 它实现的是一个范围映射到一个值上面的形式。为了便于描述，我把这种形式称作“多对一”的替换形式。

虽然使用逻辑判断进行替换操作, 实现了的“多对一”的替换形式，但是我认为仍然存在着不利于代码复用（重复使用）的问题，因此在保证代码逻辑不变的前提下，我对这段程序进行了优化，将逻辑判断也放入函数中。代码如下：

```
def age_replace(age):
    if age > 0 and age <= 6:
        return "少年"

    elif age > 7 and age <= 18:
        return  "青年"

    elif age > 19 and age <= 65:
        return "中年"

    else:
        return "老年"


print(age_replace(80))

```

复制代码

我为你解释一下为什么要把替换操作放在函数中。这样使用有两点好处：

1. 提高代码的复用，当你下次需要做年龄到年龄段映射时，可以直接调用函数，不用重复编写逻辑判断的代码。
2. 对代码进行再设计的时候，方便将逻辑判断中类似“age > 0 and age <= 6”的判断逻辑再封装成函数。

你可能会问了，代码不是应该先思考运行过程，经过设计之后再编码的吗？

在实际工作中，随着人们对代码的不断修改，原来设计好的代码结构，在整体结构上会增加很多的判断逻辑，代码质量会越来越混乱。这时候就需要你重新对代码逻辑进行优化。所以为了不让后续修改代码逻辑的行为破坏代码的可读性，就应该在初次设计和编写代码的时候考虑好代码的扩展性。

最后，我想再强调一下，使用 if 逻辑判断的目的是为了实现把一个范围映射到一个新的值，这样它就间接地实现了替换功能。所以当你解决替换问题的时候，不要把思维只局限在字符串自带的 replace() 函数中。

小结

我来给你总结一下今天的主要内容，围绕着“替换”这一功能，我给你讲解了三种实现替换的方法：

1. 字符串的 replace() 函数；

2. 使用字典做“一对一”映射，通过字典类型的键值对, 实现内容替换；

3. 使用逻辑判断实现“多对一”映射，将 if 判断的条件替换为匹配成功的结果。

替换操作要根据被替换内容的形式，选择合适的方法，replace() 函数更适合单个替换，字典适合“一对一”替换，if 逻辑判断适合将一个范围替换成一个值。

除了灵活掌握不同的替换方式，我还建议你把字典和逻辑判断放入自定义函数当中，当你遇到类似需求的时候就可以直接复用代码。

思考题

通过将城市的拼音替换成汉字的功能，你是否能实现一个自己的自动多文件标点符号替换函数，将英文符号替换为中文符号呢?

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 春节特别放送3 | 揭晓项目作业的答案

精选留言

写留言

> 由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。